CISCO SYSTEMS

Document Number	ENG-28376
Authors	Dennis Lau, Fan Jiao
Program Manager	Ulrica de Font-Menares
Status	Draft

IOS CNS Event Services Client System Software Functional Specification

Summary

The IOS CNS Event Service is a thin layer of software module in IOS Classic that allows IOS devices to publish and/or subscribe network notification events to and from the Directory. Note that this specification is a part of IOS CNS Directory Client Functional Specification

Reviewers

Name	*Department*	*Date*
Ray Bell	Cisco Network Service Engineering	
Ram Golla	Cisco Network Service Engineering	
Sunil Srivastava	Cisco Network Service Engineering	
Tony Zhang	Cisco Network Service Engineering Test	
Ulrica de Fort-Menares	Cisco IOS Infrastructure	
Dean Hiller	Cisco IOS Infrastructure	
Leo Pereira	Cisco IOS Infrastructure	
Shankar Natarajan	Cisco IOS Infrastructure	
Hugh Wong	Cisco IOS Infrastructure	

Modification History

Revision	*Date*	*Originator*	*Comments*
0.1		fjiao	Init
0.2		dwlau	Added Architecture, CLI, and Issues and Dependencies
0.7		fjiao	Inputs from team review
1.0		fjiao	Added a couple of new APIs

NOTE: This document is meant to specify and/or accompany software that is still in development. Information in this documentation may be inaccurate or may not be an accurate representation of the functionality of the final specification or software. Cisco assumes no responsibility for any damages that might occur either directly or indirectly from this documentation and/or these inaccuracies.

Table of Contents

1 Functional Overview

1.1 System Overview

The IOS CNS Client consists of a thin software component, Event Service Client (ESC), which depends on the rest of features of IOS CNS Client (LDAP V3 and Locator). ESC links network elements and directory-enabled desktop applications through use of directory technology. Monitoring the IOS device's events, and reporting the occurrences of these events may be optional because there are other means available in IOS devices to monitoring device events. ESC will be implemented as a Server and a Subsystem in IOS Classic.

Example usage scenarios:

- A network manager uses a Cisco Directory-enabled QoS application to add WFQ configuration to a backbone 7513. The event *ConfigUpdateForQoS* is processed by the CNS Event Server in a specific Domain Controller. The Event Server sends event notification (EN) to a router application that has already subscribed to the event. The notification "wakes-up" the Event Service Client (ESC), which is located on the router. In response to an Event Notification, the application binds to the appropriate Event Server using Locator API and retrieves the event(s) using CNSES API, which are wrappers over LDAP extension controls. Upon receipt of the event data, applications use a regular LDAP parse API to retrieve associated attribute value pairs.

- An event from a LS1010, *BandWidthOverSubscribed*, has been generated in an IOS application. An Event Service Client API (ESCAPI) is called to post the event to the CNS Event Services Server (ESS) . The Event Server then sends the event notifications to interesting parties, including a Policy Server.

- AS 5300 access device could generate *UserLogin* event to it's ESS, which can be consumed by applications such as 'Provisioning Server', 'Billing Application' etc.

1.2 Architecture

Devices that are running IOS, which consists of event service component, would normally locate and contact the Directory Service (DS) to retrieve event types that are available for subscription. The devices can be enabled to subscribe all event types they can support, or configured to subscribe specific events. Device profile or device family profile can specify all available events for ESC event consumer applications to subscribe.

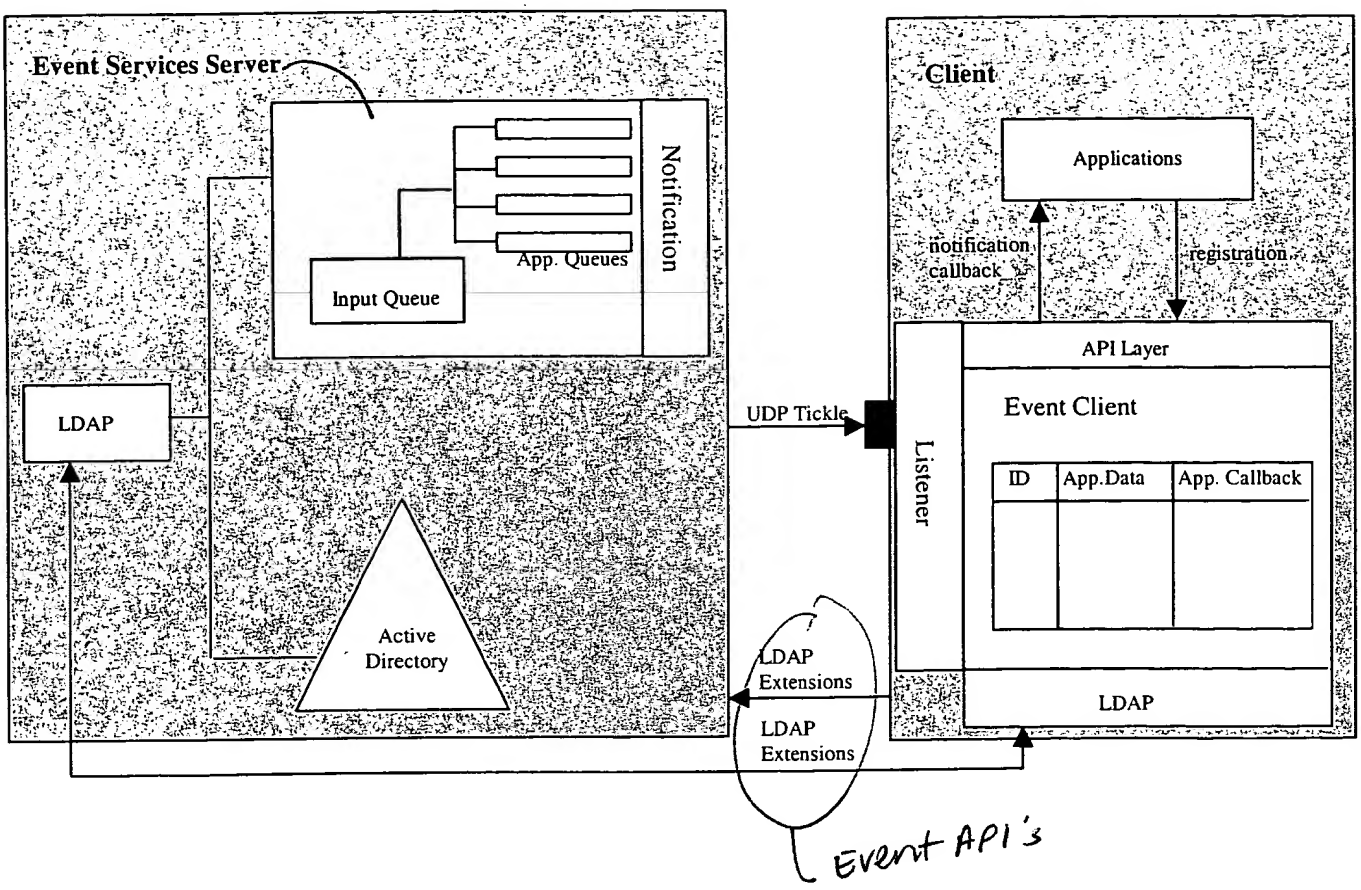
Once the device successfully finishes the event application registration, which includes registration of producer or consumer and event registration, the IOS Event Service Client process acts as a Server listening to a UDP port (which is needed for identification) and waits for notification from the Event Service Server (ESS). The notification signals the IOS ESC process that new events are available. The signal may consist of a unique consumer/producer ID so that the ESC process that acts as an event notification dispatcher will pass the notification to other IOS event applications/processes that have previously registered for receiving events.

To retrieve the new event data, the processes authenticate and LDAP binds to the ES to retrieve and process the event data. To enable multiple applications within a device to work with DS and receive events, some common functions will be provided so the applications can authenticate and bind to DS.

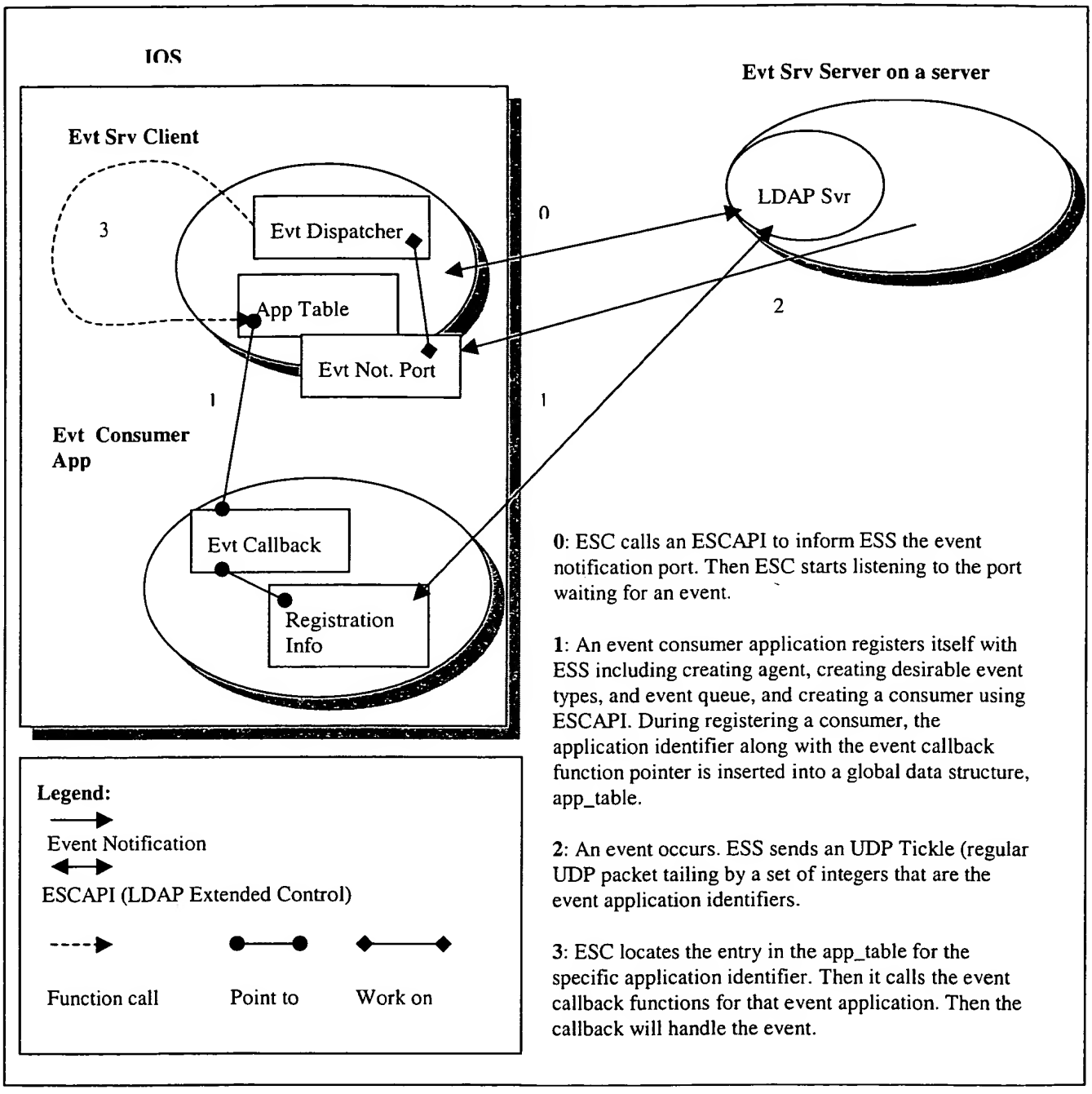
Event Services API are wrappers over LDAP extended controls.

The following figure describes the overall architecture:

1.2.1



The following diagram shows a complete sequence of execution of ESC, an event consumer application, with respect to ESS:



1.3 Feature List (Software)

The IOS CNS Event Service Client consists of a set of APIs that allows an IOS application to register itself with the Event Server, register events with ES, consume/produce an event, or unregister itself. It also has an event dispatcher that dispatches control to application callback functions that are registered by Registration APIs (see the previous diagram).

1.3.1 Event Notification

An event notification is a UDP packet whose data consists of one or more integers. These integers are unique identifiers for Consumers or Producers. Upon receipt of a notification message, ESC signals the applications with matching IDs through call back mechanism. Attaching event data along event notification through normal UDP protocol would suffer security attacks (for example, an attack on the event CHANGE_CONFIGURATION would shut down a router interface).

1.3.2 Event Service Client APIs

IOS CNS Event Service offers a set of Client APIs that makes an IOS Event Service application writer's life much easier. The APIs enable an application to be registered/unregistered with Event Service Server (ESS), as well as interesting events. As a result of application registration, the application ID and associated callback function are stored in the ESC table. The event dispatcher uses this table to dispatch control to the application's callback function. It also provides event posting and consumption APIs. The LDAP Server that implements those extended controls will call to a set of Event Service Server APIs to create and manipulate event infrastructure (event queues, etc) on the server side accordingly.

1.3.3 Event Dispatcher

Event Dispatcher listens to a UDP port (yet to be defined) for event notification. As soon as a notification arrives, the control will be dispatched to a specific callback function defined in the event table which was created during application registration.

1.4 Management Considerations

1.4.1 Command Line Interface

1.4.1.1 [no]cnses

- Enable/disable CNS ES. By default it is disabled.

1.4.1.2 show cnses [app | stats]

- Display ll applications with subscribed events

- Display statistics of total number of events received, total number of executions of each event applications, and total number of failures for each application

1.4.1.3 debug cnses [tickle | map | api | agent | event | queue | regcons | cons | prod]

- Display udp tickle info (port number, and stats)

- Display app_table
- Display all API status/return code info
- Display create_agent API info
- Display create_event API info
- Display create_queue info
- Display consume_event info
- Display post_event info
- Display consumer registration info (combination of some of the previous ones)

1.5 Testability Considerations

The test specification will be in a separate document.

1.5.1 Test for Event Dispatcher Only

This is to identify the correctness of the event dispatcher to see if all control flows are correct.

1.5.2 Test for Event Dispatcher with CLI

This is to see if CLI options are implemented correctly.

1.5.3 Test for Event Service Client APIs with one application without LDAP

This is to verify if all APIs work with stub LDAP calls (parameters are examined), and if callbacks are getting called. This is the first testing milestone to show the basic code works.

1.5.4 Test for Event Service Client APIs with one application integrated with LDAP without Server Extended Controls (Faked Server Extended Controls)

This test examines if LDAP server gets correct LDAP extended control data from client. This is the second testing milestone to indicate successful integration with LDAP. This will address error handling of the APIs since LDAP calls will fail.

1.5.5 Test for Event Service Client APIs with one application with Server Extended Controls without CLI

This test verifies that Server Extended Controls work with Event Service Client without CLIs.

1.5.6 Test for Event Service Client APIs with one application with Server Extended Controls with CLI

This test verifies that everything works. This is the third milestone (first for the system test).

1.5.7 Stress/System Tests

These types of tests are used if the device performance degrades or if the queue is too long. They can also be used to determine how many applications can run on a device without significant performance problems. These tests would provide some good pointers for final tuning of the code in terms of memory, and answer questions such as if or what we want to limit, and determining the correct limitations.

2 External Specification

2.1 Security Considerations

The communication between CNS Server and device is secure. The notification uses only one or more integers as IDs. The actual event data are obtained through a secured LDAP protocol.

2.2 Performance Considerations

It has low memory allocations. It should have good performance since the most complex operation is to decode the event data through use of LDAP V3 BER functions.

3 Internal Specifications

3.1 Event Service Client API

3.1.1 Conventions

Event Service Client API (ESCAPI) is a set of wrapper functions of LDAP extended control functions. It uses LDAP API's convention regarding to synchronized or asynchronized APIs. Only the asynchronized APIs are discussed below. There will be an extra parameter for msgid in asynchronized APIs, the result parameter will be set to message id of a request if the ldap_extended_operation suceeds.

3.1.2 Major Data Structures

3.1.2.1 Definition of Return Code

The definition contains 58 standard LDAP V3 return codes, and 21 for Event Service Client APIs.

```
#define CNSES_BAD_NOTIFICATION 1001

#define CNSES_CREATE_PROC_ERROR 1002

#define CNSES_CREATE_TIMER_ERROR 1003

#define CNSES_SERVER_ERROR 1004

#define CNSES_SERVER_EXT_CTRL_ERROR 1005

#define CNSES_REG_AGENT_ERROR 1006
```

#define CNSES_REG_PRODUCER_ERROR 1007

#define CNSES_REG_CONSUMER_ERROR 1008

#define CNSES_REG_EVENT_ERROR 1009

#define CNSES_BAD_EVENT_DATA 1010

#define CNSES_REG_QUEUE_ERROR 1011

#define CNSES_BAD_GUID 1012

#define CNSES_BAD_ID 1013

#define CNSES_CALLBACK_NULL 1014

#define CNSES_CALLBACK_ERROR 1015

#define CNSES_INSPECT_EVET_ERROR 1016

#define CNSES_POST_EVENT_ERROR 1017

#define CNSES_CONSUME_EVENT_ERROR 1018

#define CNSES_LOCK_EVENT_ERROR 1019

#define CNSES_REMOVE_AGENT_ERROR 1020

#define CNSES_PORT_NEG_ERROR 1021

#define CNSES_BAD_BER_ALLOC 1022

#define CNSES_REQUEST_EVT_CB_ERROR 1022

#define CNSES_UNLOCK_EVENT_ERROR 1023

3.1.2.2 Definition of Event Callback Function

An event callback function is defined by an Event Service Client application for handling a particular event. The function pointer is assigned to the event table entry that is passed in the API cnses_register_event. Upon receipt of a notification for an application, the event dispatcher will dispatch control to the corresponding event callback function.

```
typedef int (*cnses_callback_t) (int action,
                                 void *client_data);
```

3.1.2.3 Definition of Exit Function of Event Service Client Application

The exit function of ESC provides a convenience way for each ESC application to specify which actions are to be carried out when ESC is disabled. Usually it consists of memory de-allocations that have been claimed by the application. This will eliminate memory leaks due to possible shutdown of ESC. This function will be passed to ESC through registration of either Producer or Consumer application.

```
typedef int (*cnses_exit_callback) ();
```

3.1.2.4 Definition of Event Info

Event Info is a data structure used by Event Service Server. It is used in the Consume_Event API to return the details of the event.

```
typedef struct cnses_event_info_t_ {
    int type_id;
    int priority;
    int id;
    long time_stamp;
} cnses_event_info_t;
```

3.1.3 int cnses_get_guid (LDAP* ctext,

char* guid,

int type,

int if_persistent,

int * msgid);

Type	Name	Description
LDAP*	ctext	LDAP Context
char*	guid	the GUID that is generated in ESS. An output parameter.

int	type	{GUID_AGENT,GUID_EVENT,GUID_QUEUE,QUID_CONS, QUID_PROD}
int	if_psersistent	A "1" value means ESC recommends ESS to store it in the device profile
int*	msgid	output parameter set to message id of request if ldap_ext_op suceeds

Description: For a detailed description refer to Section 1.6 of *ESS Tutorial*. The API may be called each time to create any objects in t his event system since all objects including agent, event, producer, queue, and consumer all need to have their GUIDs.

Action: Calls a ldap_extended_op to generate a GUID in ESS. ESS may store it in some schema such as device profile.

Postcondition: The output parameter guid is updated to have a globally unique identifier.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_BAD_GUID, or 0 (success)

3.1.4 int cnses_register_agent (LDAP * ctext,

char * agent_guid,

char * agent_desc,

int agent_min_level,

int agent_max_level,

int * agent_id,

int * msgid);

Type	Name	Description
LDAP *	ctext	LDAP Context
char*	agent_guid	Agent's unique identification GUID, cnses_get_guid will generate a new GUID.
char *	agent_desc	Human readable description string for the agent.
Int	agent_min_level	Minimum version level agent can handle.
Int	agent_max_level	Maximum version level agent can handle.
int *	agent_id	output parameter, ESS generated agent id
int *	msgid	the same

Description: An Event Service Client application can only create exactly one (1) agent for a producer or consumer application. An Agent is the runtime entity of an application. This API may be called only by Event Dispatcher as an application, but doesn't need to create a separate agent.

Precondition: ldap_init, and ldap_sasl_bind should be called _guid needs be a proper GUID. Both agent_*_level are used to negotiate supported server level with Event Service Server.

Action: An ldap_extended_op (client side extended control) is issued. Then a ldap_parse_extended_result is called to fetch the data returned by LDAP Server. An agent is created in Server. The client ip addr and tickle port num will be passed to ESS so that it can send tickle to the router identified by the ip addr, through the specific port; user application doesn't need any knowledge of this.

Postcondition: The application table gets altered (agent_id gets assigned). The global storage for the application table will be changed.

Return Codes:
CNSES_SERVER_ERROR,CNSES_SERVER_EXT_CTRL_ERROR,CNSES_REG_AGENT_ERROR, or 0.

3.1.5 int cnses_register_event (LDAP * context,

int agent_id,

uchar *evt_names,

int *evt_min_level,

int *evt_max_level,

int *evt_neg_level,

int * msgid);

Type	Name	Description
LDAP *	context	LDAP Context
Int	agent_id	Registered agent's id.
Uchar *	Event_name	Name of the event.
int *	evt_min_level	output parameter – the minimum supported version level.
int *	evt_max_level	output parameter – the maximum supported version level
int*	evt_neg_level	output parameter – the negotiated version level
int *	msgid	the same

Description: This API may be repeated called to generate all desirable event types for a given agent. One needs to register events before creating event queue. Note that priority of an event is determined during an event is being posted by a producer; it is not pre-defined while an event type is created which is essentially dealt here.

Discussion: One may not delete an event types as they are created, and populated in Directory. Directory is not a database where referential integrity is guaranteed – there is no way for you to delete an object, and trace all its references. On the other hand, one may not add a new event type into an existing queue as producers might not know the existence of the new event, and so may be the consumers. See Section 2.5 of *ESS Tutorial*.

Precondition: Event data and name should be initialized properly.

Action: An ldap_extened_op is called to register event. An event type is created.

Postcondition: Event type will be created in the directory.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_BAD_EVENT_DATA, CNSES_REG_EVENT_ERROR, or 0

3.1.6 int cnses_create_queue (LDAP * context,

int agent_id,

char* queue_guid_id

char * queue_string_id,

int evt_count,

uchar **event_names,

int * msgid);

Type	Name	Description
LDAP *	context	LDAP Context
int	agent_id	Registered agent id.
char*	queue_guid	Unique queue identifier – guid
char*	queue_string-id	Human readable string associated with the queue.
int	evt_counts	number of event names
uchar **	event_names	Null terminated array of event names
int *	msgid	the same

Description: This API may be called only once if multiple applications want to share 1 queue for the purpose of load balancing, and reduction of overhead on server. There are, however, applications that require having multiple queues (for instance, one queue may contain all high priority events, the other, medium or low priority ones). Usually each consumer application has only 1 queue.

Discussion: For detailed discussion on using 1 queue or multiple queue see Section 2.8.1 of *ESS Tutorial*.

Precondition: Queue needs a combination of a guid and a string id, number of events, and guid event list to create itself.

Action: It calls a ldap_extented_op to create a queue per agent for all events. On return with ldap_extended_results it gets queue ID. An event queue is created for the particular agent.

Postcondition: Event table gets updated w/ returned event type IDs.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_REG_QUEUE_ERROR, or 0

3.1.7 int cnses_create_producer (LDAP * context,

int agent_id,

int evt_counts,

uchar ** event_names,

cnses_exit_callback_t clean_up,

int* evt_ids,

int *producer_id,

int * msgid);

Type	Name	Description
LDAP *	context	LDAP Context
int	agent_id	agent identifier
int	evt_counts	number of events
char**	evt_names	event names
cnses_exit_callback_t	clean_up	clean up function provided by application
int*	evt_ids	output parameter – event ids
int*	prod_id	output parameter – producer id
int*	msg_id	msg id

Description: This API may be called once in an application to create a producer's identity. One cannot add or delete any events from then on.

Precondition: event names should be correct.

Action: It calls a ldap_extented_op to create a producer. On return of a ldap_extended_result is done to get all producer counters, and ProducerID. A producer is created. The client ip address as well as port number are needed to be passed w/o knowledge of application.

Postcondition: A producer is created. Each event will be translated to return its ID. The application table is updated, and the producer ID, as well as all event ids will be returned to user.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_REG_PRODUCER_ERROR, CNSES_BAD_EVENT_DATA, or 0

3.1.8 int cnses_create_consumer (LDAP * ctext,

int agent_id,

cnses_guid queue_guid,

char * queue_string_id,

int event_count,

uchar ** event_names,

cnses_callback_t callback,

void* cb_data,

cnses_exit_callback_t clean_up,

int *consumer_id,

int * msgid);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	agent_id	Agent's run time ID.
cnses_guid	queue_guid	Unique queue identifier.
char*	queue_string_id	Human readable string associated with queue, could be NULL.
int	event_count	Maximum events that can be queued.
uchar **	Event_names	Event Names.
cnses_callback_t	callback	event consumer's callback function
void*	cb_data	callback user data
cnses_exit_callback_t	clean_up	exit function provided by application
int*	consumer_id	output parameter - unique consumer ID.
int*	msgid	the same

Description: This may be called once in an application to create its identity.

Precondition: Queue id is needed (quid and string id), so is the size. Also it requires a callback function to be specified.

Action: It calls an ldap_extented_op to create a consumer. On return of another ldap_extended_result will be called to get ConsumerID back.

Postcondition: The consumer will be created with the id=consumer_id. Two lists will be returned for events. The returned var diff_count = real_event_count – outlist_size.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_REG_CONSUMER_ERROR, CNSES_CALLBACK_NULL, or 0

3.1.9 int cnses_post_event (LDAP * ctext,

int agent_id,

int producer_id,

int evt_type_id,

int evt_priority,

uchar* event_data,

int* event_id,

int * msgid);

Type	Name	Description
LDAP *	Context	LDAP Context
int	agent_id	Agent ID.
int	Producer_id	Producer ID.
int	Evt_type_id	Runtime event ID.
int	Evt_priority	Event priority.
char*	event_data	event buffer
int*	event_id	output parameter – event id
int *	msgid	the same

Description: This API may be repeated to allow post multiple events in a producer application.

Precondition: Unlike register_event, LDAPMod cannot be used here. An event is presented by its id, pri, data length, and event data.

Action: It calls a ldap_extented_op to post an event. On return of ldap_extended_result will be called to get event id back. The event data will be sent out.

Postcondition: The event is put into the event queue.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_BAD_EVENT_DATA, CNSES_POST_EVENT_ERROR, or 0

3.1.10 int cnses_request_event_callback (int consumer_id, int* m_id)

Type	Name	Description

int	consumer_id	consumer id
int*	m_id	

Description: This API must be called in event application's callback. When an event is inspected, and removed, this function will be called to re-register ESS event tickle services.

Precondition: The consumer must be created.

Action: It will make ESS to call some Cisco-added function to set up next event tickling.

Postcondition: The next tickling mechanism is set up.

Return Codes: CNSES_REQUEST_EVT_CB_ERROR, or 0

3.1.11 int cnses_consume_event (LDAP * ctext,

int app_id,

int timeout,

cnses_event_info_t* event_info,

uchar* event_data,

int * msgid);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	app_id	consumer id
int	timeout	
cnses_event_info_t*	event_info	output parameter, event meta data.
char*	event_data	event data buffer
int*	msgid	the same

Description: This may be repeatedly called for an application to consume events.

Precondition: The consumer is created and queue is created

Action: It calls a ldap_extented_op to get event data. The ldap_extended_search_result will put the event data into tbl. The event will be deleted from the queue is op_type is of *TAKEN.

Postcondition: The event data is passed back from command line, and updated in app table. It will call NextEvent in ESS.

Return Codes: CNSES_SERVER_EROOR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_INSPECT_EVENT_ERROR, CNSES_CONSUME_EVENT_ERROR, CNSES_LOCK_EVENT_ERROR, or 0

3.1.12 int cnses_inspect_event (LDAP * ctext,

int app_id,

int timeout,

cnses_event_info_t* event_info,

uchar* event_data,

int * msgid);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	app_id	consumer id
int	timeout	
cnses_event_info_t*	event_info	output parameter, event meta data.
char*	event_data	event data buffer
int*	msgid	the same

Description: This may be repeatedly called for an application to peek a locked event

Precondition: The consumer and queue are created, the event is locked.

Action: It calls a ldap_extented_op to get event data. The ldap_extended_search_result will put the event data into tbl. The event will be deleted from the queue is op_type is of *TAKEN.

Postcondition: The event data is passed back from command line, and updated in app table. It will call NextEvent in ESS.

Return Codes: CNSES_SERVER_EROOR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_INSPECT_EVENT_ERROR, CNSES_CONSUME_EVENT_ERROR, CNSES_LOCK_EVENT_ERROR, or 0

3.1.13 int cnses_lock_event (LDAP * ctext, int consumer_id, int msec, int* m_id);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	consumer_id	consumer id
int	msec	msecs
int*	m_id	message id

Description: This may be called to lock the next event for peeking (cnses_consume_event with op=PEEK).

Precondition: The consumer must be created.

Action: It will make ESS to call ICDSESConsumerQueueConnection_LockNextEvent.

Postcondition: The event is locked

Return Codes: CNSES_LOCK_EVENT_ERROR or 0

3.1.14 int cnses_unlock_event (LDAP * ctext, int consumer_id, int msec, int* m_id);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	consumer_id	consumer id
int*	m_id	message id

Description: This may be called to unlock the pending. It should be used with pairing with cnses_lock_next_event.

Precondition: The consumer must be created.

Action: It will make ESS to call ICDSESConsumerQueueConnection_CancelEventRequest

Postcondition: The pending event is unlocked.

Return Codes: CNSES_UNLOCK_EVENT_ERROR

3.1.15 int cnses_remove_event (LDAP * ctext, int consumer_id, int msec, int* m_id);

Type	Name	Description
LDAP *	Ctext	LDAP Context
int	consumer_id	consumer id
int*	m_id	message id

Description: This may be called to remove the next

Precondition: The consumer must be created.

Action: It will make ESS to call ICDSESConsumerQueueConnection_RemoveEvent

Postcondition: The pending event is unlocked.

Return Codes: CNSES_REMOVE_EVENT_ERROR, or 0

3.1.16 int cnses_unregister_agent (LDAP * ctext, int agent_id);

Description: This API should be called when some exceptions occur.

Precondition:

Action: It calls a ldap_extented_op to delete the agent so that all events will be removed, all applications (producers and consumers), event queue and agent will be destroyed.

Postcondition: no outputs.

Return Codes: CNSES_SERVER_ERROR, CNSES_SERVER_EXT_CTRL_ERROR, CNSES_REMOVE_AGENT_ERROR, or 0

3.1.17 Discussions:

3.1.17.1 Event Service Client Shutdown

Event Service Client is a server in an IOS device. It is enabled/disabled through CLI. Upon exit of ESC, it is important to de-allocate all memory allocated not only by ESC applications, but also allocated in ESS. It was why the exit callback function cnses_exit_callback_t was introduced. De-allocating ESS's memory is carried out by ESC to un-register each agent created by ESC application.

3.1.17.2 Server Down

Server going down is an interesting situation. There will be different impact depend on if one is a consumer, producer or server itself. Some contexts in Section 2.9 of ESS Tutorial have good discussions on re-establishing lost connections.

3.1.17.2.1 Consumer Client

Consumer clients cannot distinguish between no events and the fact that server is down. The only possible problem would be that consumers would be overrun if server is restarted and if there are many events recovered by server. A possible solution would be to implement some delay mechanism to block itself if it determines too many events coming. Such mechanism would result in giving up process resource so that other processes can get the time slice to run.

Consumers can always re-register themselves with another server if there is such notification generated.

3.1.17.2.2 Producer Client

Producer clients not only care about server going down, but also are able to discover the event. Posting event should result in returning of an error code. Optionally it would get QueueStatus by issuing ICDSESQueueManager_GetQueueStatus to get some dynamic aspects of the queue. It server is down then an error code should be returned. If server is in bad state, but yet to be able to finish the function call, then returned statistics would provide some good clue regarding event queue, and the server itself.

In order to be able to re-register themselves with another server, producers should be also consumers of some system events.

3.1.17.2.3 Server

The fault tolerance feature of ESS is mainly depend on restart, whereas its per-event logging will act as a session manager to ensure to start from where it is left off. Currently there is no ESS-to-ESS heart beat protocol implemented. That is, an ESS doesn't know when one of its neighbor goes down. Furthermore even it knows it cannot re-designate itself as the replacement server as although event types are replicated, event queue, producer and consumer info are not in Directory.

Suppose they were (there would be a server-server heart beat protocol, and event queues, consumers, and producers are replicated in Directory, and producers are also consumers listening to some important system event). Then it would be possible for an ESS to discover the dead ESS. Since events, queues, consumers, and producers are replicated in Directory, the ESS would elect itself as the replacement server, and send notifications to both consumers and producers (in fact it becomes a producer). Consumers would simple to register themselves to the replacement server. For producers, its consumer body will inform them to re-register themselves with the replacement server.

In short there are three (3) conditions that must meet in order to have the proposed mechanism for real fault tolerance:

3.1.17.2.3.1 Producers should also be consumer of some system events

3.1.17.2.3.2 There should be a ESS->ESS heart beat protocol and a elect process to allow an ESS to discover dead ESS and elect itself or someone else to be the replacement server

3.1.17.2.3.3 Both event queues, consumers and producers should be replicated in Directory

3.1.17.3 Server Restart Notification Support

A client polling a server to see if the server is alive is not scalable. Therefore, an event, called Restart Event is one of the default events that a consumer must subscribe to. In this way, it will handle the same way as any other events.

3.1.17.4 Client Keep-Alive Protocol

Allowing thousands of routers to poll many servers may cause a scalability problem on the Internet. How much damage would be done if a server is dead, and many clients don't have notifications? Would it make sense for a router to periodically poll a server to find out if the server is dead? Would it make sense for a server to implement a similar heart beat protocol to find out if a server goes down? We believe client side's keep-alive mechanism would not be necessary if there is a solution on server side for server to discover dead servers and to select new server to let clients originally registered with dead server to re-register themselves with a newly selected server.

3.1.17.5 Retry Consideration for Using API

It is important to implement a retry mechanism while using this API. One might provide not only simple retries (say three tries), but also some kind of delayed retry follows failure of simple retry. For example, a second retry is delayed after 10 minutes, and a third might be delayed after one hour. The timing should be configurable. The retry mechanism is application dependent; we will let the application writer address this in the application code.

3.2 Event Dispatcher

The Event Dispatcher consists of logic of a background IOS server process, some initialization, command line interface handling, and exception handling.

3.2.1 Server Process

The server process is created as a normal IOS IP server. It listens to a UDP port (yet to be defined), and dispatches control to a registered application. The wire format is of a UDP packet whose value is one (1) or more integer. The server will use current integer as the key to search the application entry in the application table. If there is a match found then it will call either all registered event callback functions, or it will issue another LDAP extended control to

get the occurred event id and then call the exact event callback function. It will listen to a port that is known by ESS during initialization.

In initialization, the server process may setup some exception handling for some system signals.

3.2.2 Initialization

There is an IOS subsystem entry point that a function can call to do IOS registration, and start the server process. It will make a socket request, then store away the port number. Then it will make a call ldap_extended_operation to send the port number over to ESS. ESS will save the port number and use it to send event notifications to clients.

3.2.3 Command Line Interface Handlers

There will be routines to handle parsing the partial commands suggested in the section 1.4 of this documentation, and displaying event information as part of IOS Show Command. Both functions will be called by other IOS functions.

3.2.4 Exception Handling

There will be a set of routines to handle different system errors such as SIGTERM.

4 Issues and Dependencies

4.1 Dependencies

It depends on LDAP V3 (and also IOS credential processing for use of ldap_sacl_bind). Any IOS ES application will use the Event Service Client APIs defined in the previous section.

It also depends on CNS Event Server LDAP Extended Control library that defines and implements the server side extended controls.

4.2 Open Issues

None.

5 Reference

The C LDAP Application Programming Interface (draft-ietf-ldapext-ldap-c-api-00.txt)

SwitchSoft's Cisco Network Service Event Services Tutorial

IOS CNS/AD Client System Functional Spec (eng-25670)

SwitchSoft's Cisco Network Service Event Services Functional and Design Specification (Version 1.2)

6 Specification Review

CNS IOS Event Service Client System Functional Specification Review Meeting Minutes eng-28479)

CISCO SYSTEMS

Document Number	ENG-23055
Revision	6.0
Author	Ulrica de Fort-Menares
Project Manager	Ulrica de Fort-Menares
Commit Status	Committed

IOS CNS/AD Client

Program Plan

Project Headline

This document specifies the activities and milestones for implementing the Cisco Networking Services client component on IOS platforms as part of the Cisco's Directory Services product, scheduled for the IOS 12.0(4)T commit.

Reviewers

Department	Name and Title
Development Engineering	Anson Chen, Director IOS Infrastructure Ray Bell, Director Directory Services, NSMBU John Strassner, Chief Architect NSMBU Ram Golla, Manager Directory Services NSMBU Dalen Bosteder, Manager IP ISU Dalia Geller, Manager Security ISU
Product Mktg	Kurt Dahm, NSMBU Roger Wood, NSMBU
Program Management	Dave Berry, Program Manager NSMBU Ginny Vincenzini, Program Manager NSMBU
Customer Advocacy	N/A

Modification History

Rev	Date	Originator	Comment
1.0		Ulrica de Fort-Menares	Initial Release
2.0		Ulrica de Fort-Menares	Added comments from Ray Bell, Fan Jiao, etc. plus modification on dates
3.0		Ulrica de Fort-Menares	Added comments from Dean Hiller, changed from CDS to CNS, and modified the Program Risks section.
4.0		Ulrica de Fort-Menares	Added comments from the program plan review 7/9/98.
5.0		Ulrica de Fort-Menares	Final Draft - added the Event Services component.
6.0		Ulrica de Fort-Menares	Updated the schedule
7.0		Ulrica de Fort-Menares	Remove security

Definitions

This section defines words, acronyms, and actions which may not be readily understood.

AD	Microsoft's Active Directory
API	Application Program Interface
APPN	Advanced Peer-to-Peer Network
ARF	Automated Regression Factory
CDSI	Cisco Directory Services Interface
CNS	Cisco Networking Services
CORBA	Common Object Request Broker Architecture
DLSw	Data Link Switching
FCS	First Customer Ship
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
ENA	Extended Network Architecture
GSS-API	Generic Security Service Application Program Interface
IOS	Internetwork Operating System
LDAP	Lightweight Directory Access Protocol
PAC	Privilege Attribute Certificate
SASL	Simple Authentication and Security Layer
SDK	Software Development Kit
SSL	Secure Socket Layer
SSPI	Security Support Provider Interface
VOB	Versioned Object Base

1.0 Abstract

Rapid Internet growth has created the need for more robust, scalable and secure directory services. Cisco, through a strategic development relationship with Microsoft, has chosen Microsoft's Active Directory as the foundation of the Cisco Directory Services product. Cisco Directory Services is focused on creating rich network management and provisioning services. Cisco is committed to enabling applications to leverage advanced network services using the directory in response to the needs of business-critical applications.

The Cisco Directory Services product consists of Active Directory on Windows NT and Unix, and CNS client interfaces on Windows NT, Unix and IOS. The development effort includes:

1) porting of the Active Directory to Unix platforms such as Sun Solaris and HP-UX. Extending Active Directory's base schema to represent and manage network elements and services. Implementing Event services, security, etc.

2) creating a Software Developer Kit (SDK) for Unix and Windows NT platforms so that network-aware applications and tools can be developed.

3) implementing CDSI (Cisco Directory Services Interface) on Unix, Windows NT and IOS.

This program plan describes the development effort for the IOS CNS client. For development efforts for the Server and other Client platforms, refer to the reference section at the end of this program plan for details.

IOS CNS client consists of a number of components. Figure 1 shows the different components and their interfaces to IOS.

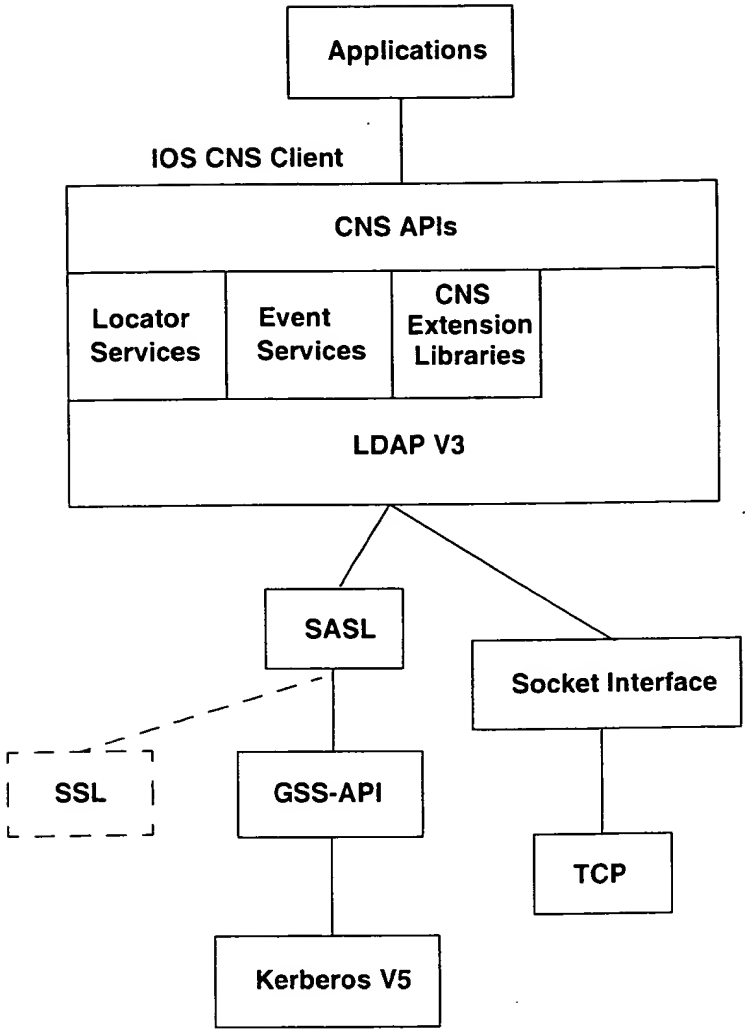


Figure 1 IOS CNS Client Components

The IOS CNS client project is a standard IOS infrastructure project that provides the infrastructure for IOS applications to query and access data that resides in a Directory Server via Lightweight Directory Access Protocol. This project is not visible as a feature. IOS applications would have to be developed in order to take advantage of the Cisco Directory Services. Some of the IOS applications that will make use of this services are:

- Network Management (eg. Auto-configuration of the router)
- QOS Management
- DHCP
- DNS
- Security (security authentication, certification administration, eg. IPSec)
- H323 applications (Call Dial plan)
- DLSw & APPN (Address caching and resource registration)
- CORBA (Naming Services and Trader services)

- Mustang (Dialer information for large scale call out)
- Cable modems

LDAP is an Internet standard that provides read and update access to directories. The standard is described in RFC 2251. LDAP uses TCP as the underlying transport and it uses the IOS socket interface.

Security is essential for controlling access to the directory servers. LDAP V3 provides simple authentication using a cleartext password as well as any Simple Authentication and Security Layer (SASL) mechanism. SASL is a layered architecture for using different security providers. GSS-API is the principal provider. Use of cleartext password is discouraged as it cannot guarantee confidentiality. In our implementation, Kerberos V5 is the default security protocol for LDAP using the IOS GSS-API as the security interface. Security technologies are changing rapidly. As other security protocols evolve, such as public-key certificates, SSL may be considered for future releases.

The LDAP Client V3 code is based on Microsoft's source code. Porting of this code is a key part of this project. In addition, there are a number of Cisco enhancements. They are Locator Services, CNS extension APIs and Event Services.

Locator Services provides a scalable solution for an LDAP client to locate the closest Directory Server in the network. Microsoft provides a rather inefficient mechanism that is administrative intensive for locating the closest Directory Server in the network. Cisco uses the sophisticated DistributedDirector which leverages the intelligence in the network to automatically, dynamically, and efficiently pick the best server for the client.

CNS Extension libraries are a set of libraries that map Cisco specific schema information to octet strings, and vice versa. A set of Unicode conversion libraries will be provided to convert UTF strings to Unicode and vice versa.

Event Services is a CNS component that provides a mechanism for a directory client to register to its server events that are of interest to him. When the event occurs, the directory server notifies the clients so that the client can take appropriate actions.

2.0 Objectives

2.1 Project Priorities

The objective of this project is to deliver the CNS client capability on the Cisco platforms by Q1/99 as part of the Cisco Directory Services product.

2.2 Production Standard Costs:

N/A

2.3 Production Forecast

N/A

2.4 Software Memory Estimates

100K.

2.5 Hardware Memory Options

N/A

2.6 Key project deliverables

- Deliver IOS image on all IOS platforms. Subsets will be available for platforms with memory constraints. The subsets are synchronous vs asynchronous, extension vs. non-extension API's.
- Deliver the IOS Programmer's Guide for CNS.

2.7 Key Features

Provide an interface, namely CDSI, for IOS developers to develop applications that can access the Directory Server to retrieve network services information.

2.8 Features not supported

Security will be implemented as a featurette.

Unicode is not supported in this release.

CNS Extensions API will not be supported in this release.

2.9 Performance

To be determined.

3.0 Development Approach

To deliver a fully functional directory client on IOS in a timely manner, Cisco has licensed the LDAP V3 client code from Microsoft as part of the strategic partner relationship with Microsoft. The integration of this code to the IOS environment will be performed by the IOS Middleware development team. The IOS Middleware team will be working very closely with the Cisco Directory Services team from NSMBU.

The LDAP Client code will continue to be enhanced jointly by Microsoft and Cisco.

Other components that make up IOS CNS will be jointly developed by the IOS Middleware development team and the NSMBU Directory services development team.

Microsoft will provide ongoing software maintenance to Cisco. The IOS Middleware team will merge the fixes to IOS as appropriate.

IOS CNS can be developed in parallel for both IOS Classic and IOS ENA, the APPN development methodology will be used. This involves keeping the CNS source code in a separate repository and committing objects to the IOS repository.

4.0 Product Development Activities Entry/Exit Criteria

4.1 Strategy & Planning Phase Criteria

This was performed as part of the NSMBU Directory Services Program.

4.2 Execution Phase Criteria

4.2.1 Design Criteria

- Reviewers and impacted groups notified (Entry)
- Email alias set up for cross BU project team communications (Entry)

- System Functional and Design Specifications approved (Exit)

4.2.2 Product Implementation/Coding Criteria

- Design validated (Entry)
- Unit Testing complete (Exit)
- Code Reviews complete (Exit)
- System Functional and Design specification reviewed (Exit)

4.2.3 Internal Verification Entry Criteria

- Test Plans reviewed/approved (Entry)
- Test Designs complete (Entry)
- Review of Unit Test Results (Entry)
- Test coverage and results reviewed (Exit)
- System Integration Test with NSMBU complete (Exit)

4.2.4 Code Commit to Mainline Criteria

- Zero unresolved sev 1 & 2 defects (Entry).
- Test suites automated (Entry)

4.2.5 External Validation Criteria

- External test plans reviewed and approved (Entry)
- Feature and Integration Tests complete (Entry)
- Regression Test complete (Exit)
- Zero unresolved sev 1 & 2 defects (Entry/Exit)
- Automated test scripts submitted to ARF (Exit)
- CNS Programmer's API Guide for IOS complete (Exit)

4.3 Deployment Phase

- FCS Release criteria met (Entry)- this is the default
- Post Project Assessment meeting (Entry + 3months)

5.0 Key Development Tasks:

5.1 H/W Development Tasks

N/A

5.2 S/W Development Tasks

5.2.1 Directory Services VOB

Set up a new VOB for the CNS source code. This will provide a separate clearcase repository for the CNS source code. Since there is the licensing agreement between Microsoft and Cisco, access to the source code should strictly be granted to the IOS Middleware Directory Services development team only.

5.2.2 Code Drop Process

Set up a process for managing code drops from CNS Engineering for ongoing maintenance purpose.

5.2.3 Process for the IOS Build

This involves establishing a procedure with the IOS build group for the build process.

5.2.4 Integrating the LDAP V3 code

Integrate the Microsoft NT Version 5 LDAP V3 client code to IOS. Identify the subsets of the LDAP APIs that will be supported on IOS, and which platforms will use a subset on combination of subsets.

5.2.5 CNS Extension API

This includes providing a set of libraries to parse octet strings. The design for the CNS Extension API is underway by NSMBU. The work on IOS CNS Extension API will be scheduled based on the dates provided by NSMBU.

5.2.6 Kerberos Security Interface

The Microsoft LDAP V3 source code uses SSPI as the interface for network security services. IOS will be using GSS-API. This includes mapping of the Microsoft's SSPI API to the Cisco'S IOS GSS-API. (This feature will be implemented as a featurette).

5.2.7 Interface to Socket

This involves mapping of the Microsoft socket API to the Cisco IOS Socket API.

5.2.8 Locator Services

The Locator Services client will use the IOS DistributedDirector to locate the closest Directory server in the network. If the DistributedDirector is not used in the customer's network, an alternate method will be available for clients to locate the best server in the network.

5.2.9 Event Services

There are three components that are required, Event Services API for the user application, the transport and the Event Services engine. The LDAP protocol will be extended as the transport for the Event Services.

5.3 Network Management Tasks

5.3.1 SNMP Enhancements

N/A

5.3.2 MIB Enhancements

N/A

5.4 Test Engineering Tasks

- Regression test the applications that use the LDAP V2 API. Use the LDAP V2 testbed.
- Automate the test scripts and submit to ARF.
- Enhance the Cisco LDAP V2 test scripts to support LDAP V3.
- Integration testing with all of Cisco's CNS Server platforms to ensure interoperability. Use NSMBU's testbed where possible.
- Performance and stress test the IOS CNS client code.
- Interoperability testing with an LDAP V3 Directory Server.

5.5 Diagnostic Development tasks

N/A

5.6 Mechanical/Power Development Tasks

N/A

5.7 Compliance Test Tasks

5.7.1 Agency Regulatory Approvals

- Compliance testing for the regulatory approvals.

Regulatory Approval	Requirement
PTT/Network Certification	none
Safety	none
Emissions	none
EMC	none

5.7.2 Standards

- Verification testing for the standards compliance
- PTT/Network Certification shall include harmonized standards, Common Technical Requirements (CTRs) and any other means to be used in achieving compliance with the requirements of the European Directive 91/263/EEC
- The PTT/Network Certification department is responsible for determination of the appropriate PTT/Network Certification test requirements. In particular, the BABT Approvals Liaison Engineer (ALE)/Deputy Approvals Liaison Engineer(DALE) in the PTT/Network Certification department are responsible for authorizing these requirements.

Standards	Requirement/ revision level
CCITT / ITU	none

Standards	Requirement/ revision level
ANSI	none
IETF	none
RFC	RFC2251 - LDAP V3 Protocol Specification RFC2252 - LDAP V3 Attribute Syntax Definitions RFC2253 - LDAP V3 UTF-8 String Representation of Distinguished Names RFC2254 - LDAP V3 String Representation of LDAP Search Filters RFC2255 - LDAP V3 URL Format RFC2256 - LDAP V3 User Schema Summary
ETSI	none
BELLCORE	none
CTRs *	none
NETs *	none

* The PTT/ Network Certification department will supply the appropriate CTR and NET Standards.

6.0 Program Risks and Interdependencies:

6.1 Resource Contentions:

An additional resource is required for this project. No test resources assigned for the Event Services component.

6.2 External Dependencies:

CNS depends on a number of Kerberos enhancements implemented by the IOS Security Engineering group. They are the PAC format enhancements, TCP as an optional transport, remote downloading of the SRVTAB file, and the implementation of the GSS-API layer for CNS.

Note: the GSS-API is a critical component for this project.

The use of the DistributedDirector to locate the best server also depends on an enhancement implemented by the IOS Protocols Engineering group. The DistributedDirector has to support the location of the server for a specific protocol and domain as described in RFC2052. The DNS RR type is SRV RR.

6.3 Technological Risks:

Stability of the Microsoft code.

6.4 Other Risks:

The current IOS LDAP V2 code is approximately 20K. The initial code size for LDAP V3 is approximately 100K. Every effort will be made to reduce the code size, but it is not known at the time of writing how much it can be reduced. This may be a major issue for low end platforms.

7.0 Exceptions to Development Methodology

No PRD for this project.

Since there are no applications yet developed, it is difficult to find customers willing to test code which offers no new features. We will make every effort to get as wide exposure as possible, via internal EFT and monthly ARF runs. At the same time, we will make every effort to find applications that want to use the CDSI interface to leverage network services on IOS.

8.0 Resource Requirements & NRE:

8.1 Engineering Staffing

Name	Function
Ulrica de Fort-Menares	Project Manager
Gene Peterson	Target Release Program Manager
N/A	Hardware Manager
N/A	Hardware engineer
N/A	Product support engineer
Ulrica de Fort-Menares (IOS Infra) Dalia Geller (IOS Security) Ram Golla (NSMBU)	Software Manager
Sri Gundavelli (NSMBU) Dean Hiller/TBH (IOS Infra) Fan Jiao (NSMBU) Dennis Lau (NSMBU) Allen Long (IOS Security) Leo Pereira (IOS Infra) Johathan Trostle (NSMBU)	Software Engineer
Don Wooton (NSMBU)	Dev Test Manager
Tony Zhang (NSMBU) Hugh Wong (IOS Infra)	Software Engineer

8.2 Non-Engineering Project Staffing

Name	Function
Ginny Vincenzini (NSMBU)	CNS Program Manager
Kurt Dahm (NSMBU) Roger Wood (NSMBU)	Marketing
Dave Cavanaugh (IOS) Andrew Vernon (IOS)	SW Documentation contact
N/A	HW Documentation contact
N/A	CE contacts

8.3 Engineering Expenses Summary:

Summary of engineering development costs:

Development Expense	Cost
Prototypes	N/A
Support equipment expenses	N/A
Capital, test equipment	$45,000
Capital, systems	N/A
Agency	N/A
Outside services	N/A
Total	$64,000

Provide detailed tables as required. The larger the expenses the more details required.

8.3.1 Equipment costs

Equipment List	Qty	Cost	Use and disposition
Routers			
C7200	2	10K	
C7500	1	7K	
C4700	1	3K	DRP agent
C2500	1	2K	DistributedDirector & DRP agent
End Devices			
Pentium II PC's	3	12K	NT Active Directory Servers (2) & NT Directory Client
Sun Ultra 10	1	10K	Automation Server
Other Lab Infrastructure Equipments			
Sniffer	1	20K	with Ethernet/FE/ISDN/ Analog/Serial support

9.0 Schedule: Note: By definition, Milestones are the end dates.

MILESTONES	ORIGINAL PLAN	CURRENT FORECAST	COMMENTS
Strategy and Planning Phase			
t_0 First Resources Assigned			
System Functional Spec			draft available but design has not been finalised. Spec is still being modified.
Program Plan			complete - modification is made as the design changes.
t_{commit} Planning Phase Complete (Project Committed) - approves Program Plan and System Functional Spec			
Execution Phase - Software			
Software Unit Design Specification Review Complete			Spec review scheduled for
Code Complete			
Unit Testing Complete			
Integrate to IOS			
Security API Code Complete			Date to be provided by the IOS Security group
Unit Testing IOS CNS Security Interface			depending on the above date
DD Enhancement Code Complete			Scheduled for 12.0(3)T
Clearcase VOB Setup complete			Complete
Build process Setup complete			

MILESTONES	ORIGINAL PLAN	CURRENT FORECAST	COMMENTS
Internal Verification			
Test programs Complete			Event Services test resources not yet available
Test program Unit Testing Complete			
Test Plan Review Complete			
Test Plan Execution			
Test Coverage and Results Review			
Automated Test Scripts Complete			
Test Scripts submitted to ARF			
External Validation			
CNS Programmer's API Guide Review Complete	TBD		Date to be provided by the Engineering Training group
External Test Plan Review	N/A		
Begin Early Field Test	N/A		
Begin Beta Test	N/A		
FCS Readiness Review	N/A		
Deployment Phase			
CE Training	N/A		
FCS -	N/A		
General Availability	N/A		
Post Project Assessment	N/A		

References

[1] 'Cisco Directory Services Interfaces (CDSI) Functional Overview', John Strassner & Dave Berry, DSENG-021.1

[2] 'CDS Client for IOS System Functional Specification', Ram Golla, DSENG-009.7

[3] 'Cisco QoS Configurations & CDS Qos Schema System Functional Specification', Fan Jiao, DSENG-015.101

[4] 'Cisco Directory Services (CDS) Schema Specification', John Strassner, DSENG-010.4

[5] 'Cisco Directory Services Policy and Event Services Dictionary', John Strassner, J. J Ekstrom, Thomas McNeill, DSENG-011.001

[6] 'Cisco Directory Services Policy and Event Services Engine Requirements Specification', John Strassner, DSENG-012.001

[7] 'CDS Policy and Event Services Engine System Functional Specification', Thomas McNeill, DSENG-019.4

[8] 'Distributed Director: Software Unit Functional Specification', Paul Traina, Richard Johnson and Dhaval Shah, ENG-7562

[9] 'Cisco DistributedDirector', Keven Delgadillo, http://www.cisco.com/warp/public/751/distdir/dd_wp.htm

[10] 'IOS CNS/AD Client System Functional Specification', Ulrica de Fort-Menares & Ram Golla, ENG-25670

[11] 'Locator Services Functional Spec', Sri Gundavelli, ENG-28610

[12] 'IOS CNS Client Event Services System Functional Specification', Fan jiao & Dennis Lau, ENG-28376

[13] 'IOS CNS/AD Client Software Unit Design Specification', Leo Pereira, ENG-25716

[14] 'IOS CNS/AD Client Test Plan', Hugh Wong, ENG-25717

[15] 'Software Unit Design Specification for GSS-API to support LDAP V3', Allen Long, ENG-29005

[16] 'IOS CNS Client Event Services Test Plan', Tony Zhang, ENG-28950

[17] `IOS CNS Event Services Client Software Unit Design Specification', Fan Jiao, ENG-28632

Attachments

As appropriate, examples of forms, log sheets, diagrams, schematics, or other pieces of information used in or generated when carrying out the activities of the document would be identified here and then attached as appendices to the procedure. (If attachments are added, delete this paragraph.)

Project Documentation Checklist

A checklist is used to identify the documents planned for the project. This will ensure that project team members understand the level of detail expected and the phase it will be available.

1.0 Product Definition

- ___ Product Requirements Document
- _X_ System Functional Specification
- _X_ Program/Project Plan
- ___ Release Plan (for a release vehicle)
- _X_ Project Review Minutes

2.0 Design

- ___ Hardware Functional Specification(s)
- ___ System Packaging Design Specification(s)
- ___ Software Unit Functional Specification(s)
- _X_ Software Unit Design Specification(s)
- _X_ Design Review Minutes
- _X_ Project Review Minutes

3.0 Implementation

- ___ Review minutes of Implementation Phase Entry Criteria
- _X_ Feature Test Plan(s)
- ___ Integration Test Plan
- ___ Design Validation Test Plan (DVT)
- ___ System Test Plan
- _X_ Code Review Minutes
- _X_ Project Review Minutes

4.0 Internal Verification

- ___ Review minutes of Internal Verification Entry Criteria
- _X_ Test Failure Identification and Resolution
- _X_ Test Results
- ___ EFT Plans
- ___ Beta Plans
- _X_ Project Review Minutes

5.0 External Validation

- ___ Review minutes of External Validation Entry Criteria
- ___ EFT Results
- ___ Beta Test Results
- _X_ Defect Identification and Resolution logged in DDTS

- _X_ Final Documentation (IOS Programmer's Guide)
- _X_ Release Notes for Outstanding Defects
- ___Deviations written, approved, and Corrective Action Plans in place
- ___ FCS Readiness Review Minutes

6.0 Deployment

- ___ CE Training Complete
- ___ First Article Verified
- ___ ECO completed
- _X_ Post Project Assessment

Engineering R&D Review Template

1. Abstract

Taken from the original abstract.

2. Issues and Risks

New issues and risks that have become apparent since the last review.

3. Accomplishments

Accomplishments toward the goals identified in the previous review

4. Goals for Next Review

Immediate objectives of the upcoming review period that are necessary to meet the project schedule.

5. Changes to Plan

Itemize any changes in deliverables or Entry/Exit criteria from the original project plan.

6. Schedule Update

Current status of significant schedule items.